

Funciones

Programación I

Lic. Mauro Gullino maurogullino@gmail.com

UTN FRH

Funciones

Funciones = subrutinas = procedimientos

Son programas independientes

Nos permiten dividir un problema en subproblemas

Funciones en C

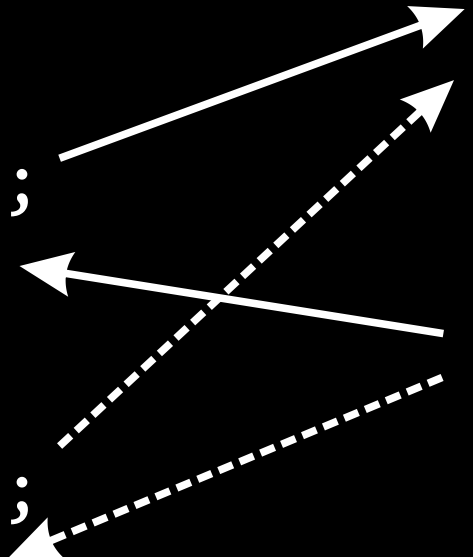
Las funciones son “llamadas” o “invocadas”

Realizan un proceso y “retornan” al “llamante”

Se produce una ruptura del flujo

Ejemplo

```
main() {  
    ...  
    printf("Hola mundo");  
    ...  
    ...  
    printf("Chau mundo");  
}
```



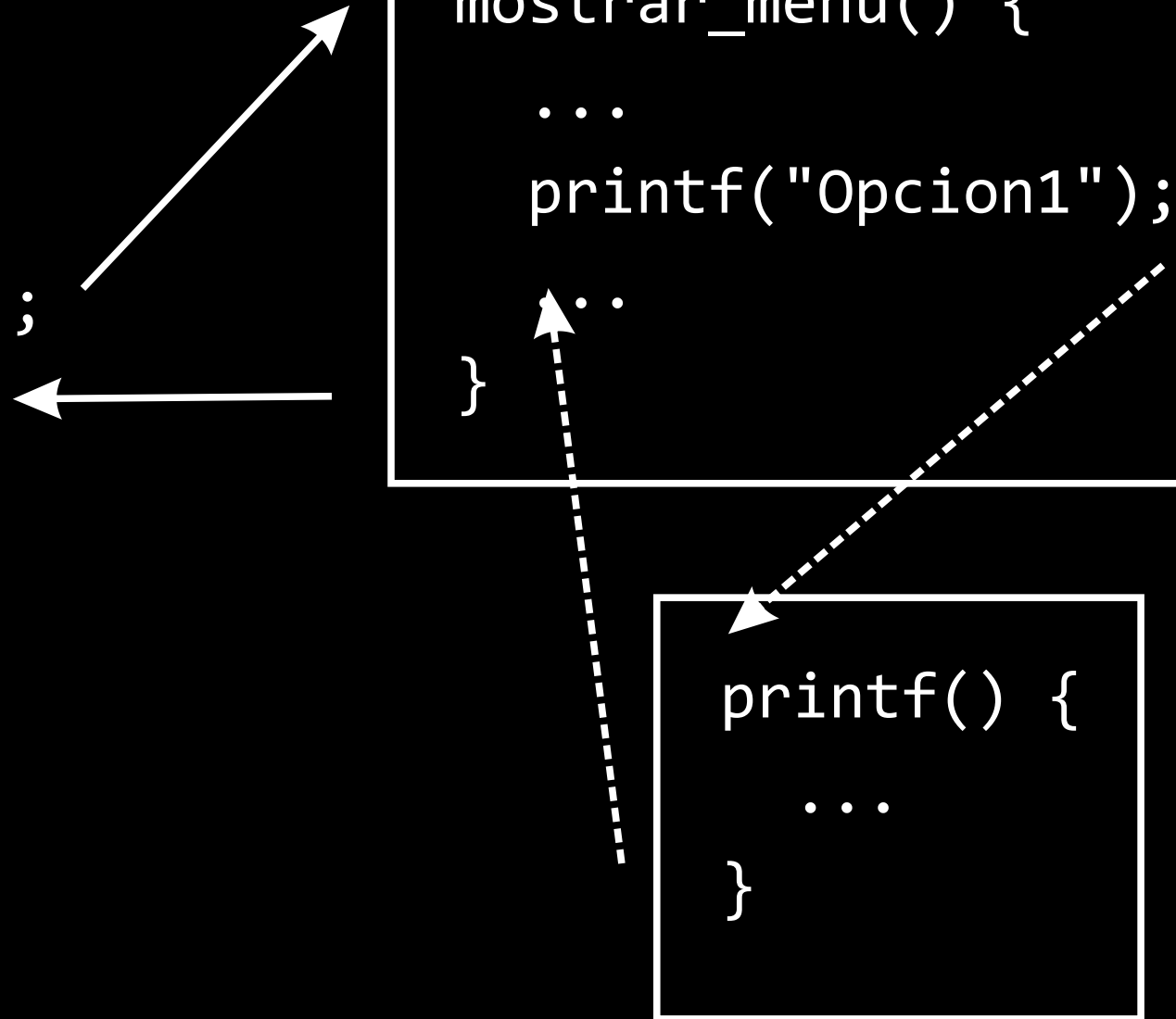
```
printf() {  
    ...  
}
```

Otro ejemplo

```
main() {  
    ...  
    mostrar_menu();  
    ...  
}
```

```
mostrar_menu() {  
    ...  
    printf("Opcion1");  
    ...  
}
```

```
printf() {  
    ...  
}
```



Características de las funciones

- a) tienen un nombre, que se usa para invocarlas
- b) pueden tener parámetros (datos de entrada)
- c) pueden retornar valores (datos de salida)



Ejemplo

```
#include <stdio.h>
```

```
main() {  
    imprime_nombre();    //qué invoca??  
}
```

```
imprime_nombre() {  
    printf("Mi nombre es Dennis");  
}
```

Ejemplo corregido

```
#include <stdio.h>
```

```
imprime_nombre();
```



prototipo de la fc.


```
main() {  
    imprime_nombre();  
}
```

```
imprime_nombre() {  
    printf("Mi nombre es Dennis");  
}
```


Ejemplo corregido 2

```
#include <stdio.h>
void imprime_nombre();
```

```
int main() {
    imprime_nombre();
}
```

 tipo de retorno

```
void imprime_nombre() {
    printf("Mi nombre es Dennis");
}
```

Pasaje de parámetros

```
#include <stdio.h>
```

```
void imprime_nacimiento(int);
```

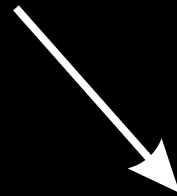


```
int main() {
```

```
    int anio = 1975;
```

```
    imprime_nacimiento(anio);
```

```
}
```



```
void imprime_nacimiento(int n) {
```

```
    printf("Ud. nació en el año %d", n);
```

```
}
```

Parámetros son variables locales

```
#include <stdio.h>
```

```
void probando(int);
```

```
int main() {
```

```
    int a = 10;
```

```
    probando(a);
```

```
    printf("a vale: %d", a);
```

```
}
```

```
void probando(int a) {
```

```
    a = a + 5; ←
```

```
}
```

Variables globales

```
#include <stdio.h>
```

```
void probando();
```

```
int a = 10; ←
```

```
int main() {
```

```
    probando();
```

```
    printf("a vale: %d", a);
```

```
}
```

```
void probando() {
```

```
    a = a + 5;
```

```
}
```

Variables locales auxiliares

```
#include <stdio.h>
```

```
void mostrar_numeros();
```

```
int main() {  
    mostrar_numeros();  
}
```

```
void mostrar_numeros() {  
    int i; ←  
    for (i=1; i<11; i++) printf("%d \n", i);  
}
```

Resumiendo

Variable	Valor inicial	Tiempo de vida
GLOBAL	cero	Todo el programa
LOCAL	basura	Función dueña
PARAMETRO	invocación	Función dueña

Retorno de valores

```
#include <stdio.h>
```

```
int anio_actual();
```

```
int main() {
```

```
    int anio = anio_actual(); ←
```

```
    printf("el año actual es %d", anio);
```

```
}
```

```
int anio_actual() {
```

```
    return 2016; ←
```

```
}
```

Retorno de valores

```
#include <stdio.h>
```

```
float promedio(float, float);
```

```
int main() {
```

```
    float nota1 = 5.5, nota2 = 6.0;
```

```
    printf("el promedio es %.2f", promedio(nota1,nota2));
```

```
}
```

```
float promedio(float x, float y) {
```

```
    return (x+y)/2;
```

```
}
```


Ejemplo de desarrollo

Objetivo

Desarrolle un programa que determine si un número entero es o no primo.

Paso 1

```
int es_divisible(int numero, int divisor) {
```

}

Paso 1

```
int es_divisible(int numero, int divisor) {  
    int resto = numero % divisor;  
  
    if (resto == 0) return 1;  
    else return 0;  
}
```

Paso 2

```
int es_primo(int numero) {
```

```
}
```

Paso 2

```
int es_primo(int numero) {  
    int i;  
  
    for (i=2; i<numero; i++) {  
        if (es_divisible(numero, i)) return 0;  
    }  
  
    return 1;  
  
}
```

Programa final

```
#include <stdio.h>
```

```
int main() {
```

```
}
```

Programa final

```
#include <stdio.h>
```

```
int main() {
```

```
    int num;
```

```
    scanf("%d", &num);
```

```
    if (es_primo(num)) printf("es primo");
```

```
    else printf("no es primo");
```

```
}
```


Objetivo

Desarrolle un programa que determine si un número entero es "perfecto", es decir, igual a la suma de sus divisores (excepto él).

6 es perfecto ($1+2+3$)

el siguiente núm. perfecto es 28

Paso 1

```
int es_divisible(int numero, int divisor) {  
    int resto = numero % divisor;  
  
    if (resto == 0) return 1;  
    else return 0;  
}
```

Paso 2

```
int suma_divisores(int numero) {
```

```
}
```

Paso 2

```
int suma_divisores(int numero) {  
    int i, acum=0;  
  
    for (i=1; i<numero; i++) {  
        if (es_divisible(numero, i)) acum=acum+i;  
    }  
  
    return acum;  
  
}
```

Paso 3

```
int es_perfecto(int numero) {
```

```
}
```

Paso 3

```
int es_perfecto(int numero) {  
  
    if(suma_divisores(numero) == numero) return 1;  
    else return 0;  
  
}
```

Programa final

```
#include <stdio.h>
```

```
int main() {
```

```
}
```

Programa final

```
#include <stdio.h>
```

```
int main() {
```

```
    int num;
```

```
    scanf("%d", &num);
```

```
    if (es_perfecto(num)) printf("es perfecto");
```

```
    else printf("no es perfecto");
```

```
}
```


Consideraciones especiales

Modos de transferencia / pasaje

- **por valor**

```
a = es_divisible(10, 3);
```

```
// parámetros serán copias de los valores
```

- **por referencia**

```
scanf("%d", &numero);
```

```
// parámetro es "referencia"
```

```
// lo que se copia es la dirección
```

Argumentos de tipo erroneo

```
#include <stdio.h>
```

```
void prueba(float a, int b) {  
    printf("%f %d", a, b);  
}
```

```
int main() {  
    int x = 4;    float y = 7.5;  
    prueba(x, y);  
}
```

Colisión local / global

```
#include <stdio.h>

void funcion();

int x = 30;

int main() {
    x++;
    funcion();
    printf("\n x en main vale %d", x);
}

void funcion() {
    int x = 20;
    printf("\n x en funcion vale %d", x);
}
```

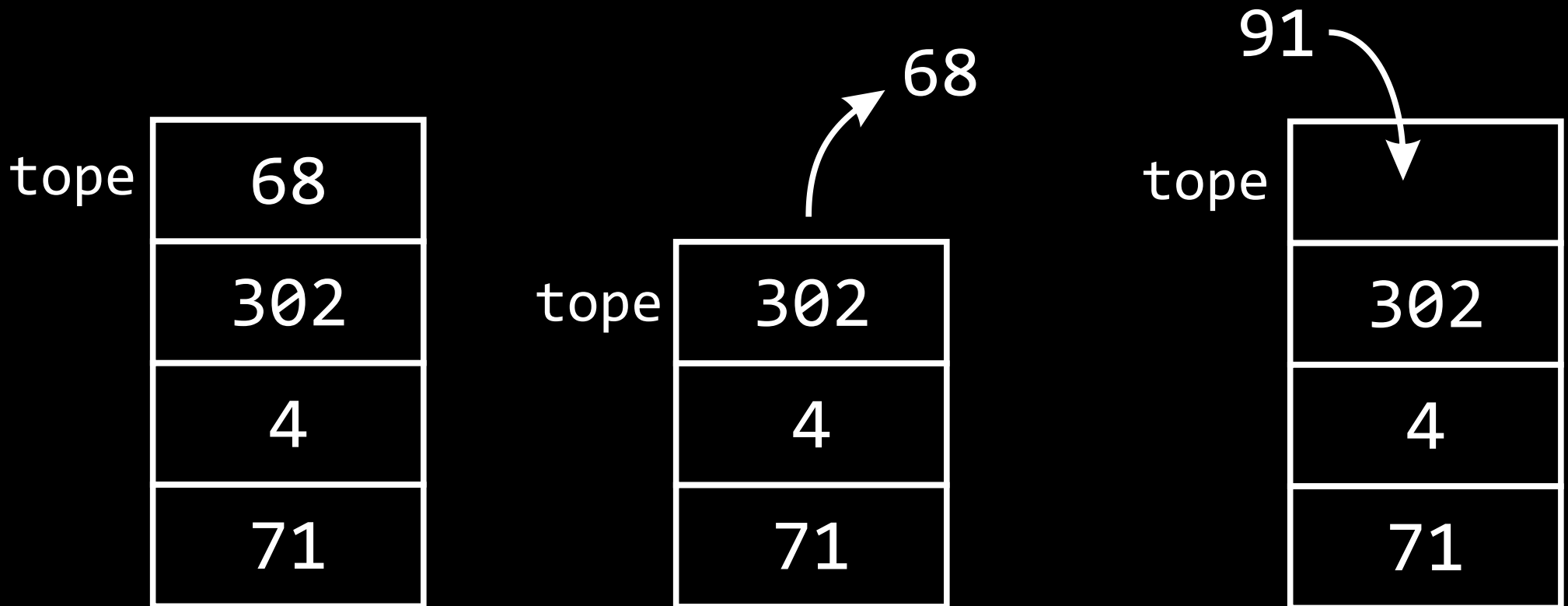
Lo único que podemos
hacer en C ...

es hacer *funciones*

Funcionamiento de la pila

Pila / stack

- Constituye una "**estructura de datos**"
- Es memoria RAM



Llamado a función

- El llamante "apila" la **dirección de retorno** y los **parámetros** para la función "a llamar"
- La función lee los **parámetros** de la pila y hace su trabajo. Para retornar lee la **dirección de retorno** de la pila y regresa a ese punto.

Ejemplo de llamadas

```
#include <stdio.h>
```

```
void prueba(char x) {  
    printf("%c", x);  
}
```

```
int main() {  
    prueba('A');  
}
```

Ejemplo de llamadas

```
#include <stdio.h>
```

```
void prueba(char x) {  
    printf("%c", x);  
}
```

```
int main() {  
    prueba('A');  
}
```

var x

65
dirección
volver

A

Ejemplo de llamadas

```
#include <stdio.h>
```

```
void prueba(char x) {  
    printf("%c", x);  
}
```

```
int main() {  
    prueba('A');  
}
```

var x

65	c %
99	
37	
dirección	
volver	A
65	
dirección	
volver	

Variables locales

- en la pila viven las variables locales
- por eso al terminar la función se "destruyen"

Recursividad

Recursividad

Es la solución de un problema utilizando funciones que **se llaman a sí mismas**.

Algunos problemas se resuelven más fácilmente con **recursividad** que con iteraciones.

Ejemplo: factorial

Calculo del factorial:

$$3! = 1 \times 2 \times 3 = 6$$

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

Podemos definir factorial así:

$$N! = N \times (N-1)!$$

El factorial de 0 es 1

Ejemplo: factorial

```
unsigned long factorial(unsigned int n) {  
    unsigned long aux=1;  
  
    if (n>0) aux = n * factorial(n-1);  
  
    return aux;  
}
```


Ejemplo: Recursividad

```
#include <stdio.h>
```

```
void funcion(int x) {  
    if(x < 10) { printf("%d",x); }  
    else {  
        printf("%d", x % 10);  
        funcion(x / 10);  
    }  
}
```

```
int main() { funcion(3487); }
```

Recursividad

La función recursiva debe tener alguna condición de ejecución que NO sea una llamada a sí misma.

De lo contrario, el programa es **infinito**:

La **memoria** se agota y el S.O. termina el proceso.
(equivalente a un bucle infinito)

Ejemplo: Recursividad 2

```
#include <stdio.h>
```

```
void funcion(int x) {  
    if(x < 2) { printf("%d",x); }  
    else {  
        funcion(x / 2);  
        printf("%d", x % 2);  
    }  
}
```

```
int main() { funcion(17); }
```